Theoretical Computer Science 411 (2010) 2223–2238

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

An exact correspondence between a typed pi-calculus and polarised proof-nets

Kohei Honda [a], Olivier Laurent [b,*]

[a] *Department of Computer Science, Queen Mary, University of London, United Kingdom*
[b] *Preuves Programmes Systèmes, CNRS, Universtité Paris 7, France*

ARTICLE INFO

Article history:
Received 27 November 2008
Received in revised form 29 September 2009
Accepted 24 January 2010
Communicated by P.-L. Curien

Keywords:
Pi-calculus
Proof-nets
Processes
Logics
Proofs
Types
Interaction
Concurrency
Linear Logic
Polarity
Embedding
Determinism
Non-determinism

ABSTRACT

This paper presents an exact correspondence in typing and dynamics between polarised linear logic and a typed π-calculus based on IO-typing. The respective incremental constraints, one on geometric structures of proof-nets and one based on types, precisely correspond to each other, leading to the exact correspondence of the respective formalisms as they appear in Olivier Laurent (2003) [27] (for proof-nets) and Kohei Honda et al. (2004) [24] (for the π-calculus).

1. Introduction

This paper presents exact mutual embeddings between the proof-nets and the π-calculus. More precisely, we show that a specific form of *polarised linear logic* [27] and a typed version of the asynchronous π-calculus [24] are essentially different ways of presenting the same structure. On the one hand, the π-calculus is a formalism which can represent a wide variety of computational phenomena starting from sequential programming languages to distributed computation through a simple operation, channel passing, originating in CCS; on the other hand, proof-nets represent dynamics of cut elimination in Linear Logic, which is a proof-theoretic reformulation of intuitionistic logic. These two formalisms come from quite different backgrounds: this paper pinpoints an exact way in which we can relate these two different formalisms.

As is well known, the so-called Curry–Howard correspondence allows us to relate proofs of intuitionistic logic to the typed λ-calculus, so that a proof corresponds to a typed λ-term, the assumptions and conclusion of a proof to the types of that term, and cut elimination (a mechanical procedure to simplify proofs preserving the same assumptions and conclusions) to

* Corresponding author. Tel.: +33 4 72 72 84 34.
E-mail address: olivier.laurent@ens-lyon.fr (O. Laurent).

0304-3975/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2010.01.028

reduction in the λ-term. Linear Logic [15] can fully embed this isomorphism: further the logic enjoys duality of the classical logic, and the notion of cut elimination in proof-nets is more fine-grained than the cut elimination in the intuitionistic logic. The connectives of Linear Logic arise as decomposition of the semantic universe underlying intuitionistic logic. The study of Linear Logic and proof-nets is usually motivated by logical or logically oriented semantic concerns, though some of the ideas in Linear Logic (such as linearity) are widely used in computational formalisms other than logically motivated ones.

In contrast, the π-calculus [35] is a relatively recent development in process algebras, syntactic formalisms for understanding concurrent computation which include, among others, CCS [32], CSP [19] and ACP [11]. In particular, all algebraic operators in the π-calculus are those of CCS: the π-calculus merely adds a syntactic treatment of name passing to CCS. The motivation to add name passing in the π-calculus is practical: the authors of [35] wish to model so-called *mobile systems*, i.e., communicating systems in which connectivity among processes dynamically changes. Such systems are commonplace nowadays, be they mobile phone networks or email communication where email addresses themselves are exchanged. The study of the π-calculus in particular and process algebras in general is motivated by modelling practical systems rather than logical concern, even though it is also used as a tool to study semantics and logics of computation.

The present work shows that, in spite of these two very different origins and orientations, there is an expressive common part shared by these two formalisms. This "common part" is obtained through a natural restriction of each formalism: proof-nets are restricted so that they are "polarised" [27], i.e. we have distinction between positive and negative formulae. For the π-calculus we use, other than asynchrony in communication, a restriction by types used for studying control operators [24]. As we shall show, the correspondence between two fragments is as exact as can be: two translations are mutually inverse (up to the structural equality of the π-calculus); the well-formed proof-nets are precisely well-typed processes via translation and vice versa; and one-step reduction in one formalism is precisely mirrored by one-step reduction in another formalism.

The result poses many questions: why one of the (arguably) most advanced tools to study dynamics of logics and one of the (arguably) most advanced tools to study dynamics of concurrent computation coincide so closely? What merits does this correspondence give us? For the former the authors do not have an answer (another related mathematical tool for computation is *game semantics* [4,18], which again has a different origin and motivation but has a close tie with both formalisms). For the second question, it may be worth recording a couple of observations we have gained through the present inquiry. First, our correspondence results treat the proof-nets which are semantically non-deterministic (in the π-calculus notation we allow typed processes of the form $!x.0|!x.0$ which introduce racing at x). While non-deterministic proof-nets have been studied before ([30] for example), the presented variant would be of interest because of its precise correspondence with the π-calculus: the whole laws and theories of non-deterministic processes from process algebras can now be applied (for example we can now discuss the notion of interactive behaviour of proof-nets directly using transition relations of the π-calculus). Second, the syntactic constructs in the π-calculus are given an exact geometric presentation through the corresponding proof-nets. While many graphical formalisms for the π-calculi have been studied [8,25,29,31], our work differs in that the target graphical formalism is directly based on proof-nets. This allows us to compare incremental geometric constraints in proof-nets on the one hand and incremental type-based constraints in the π-calculus. For emphasising this point, our presentation starts from the correspondence result for the respective non-deterministic extensions and adds incremental constraints: for each constraint we again establish one-to-one correspondence between them, finally reaching the deterministic, terminating formalisms studied in [27,24]. Thus already the both-way interactions enrich these two theories. We hope that the present inquiry serves as a starting point of further dialogues between these two different fields of study.

The connection between process algebras and Linear Logic was first observed by Samson Abramsky in his computational interpretation of Linear Logic [1,2]. The process formalism he used is not a standard process algebra but a syntactic construction which directly represents proofs of Linear Logic as terms. Bellin and Scott [10] have shown an embedding of a version of proof-nets with a notion of polarities to a "synchronous" π-calculus, a non-standard version of the π-calculus with commutativity of prefixes. Laneve, Parrow and Victor [29] showed that some parts of dynamics of the original proof-nets can be captured by the fusion calculus, which is a variant of the π-calculus with (what corresponds to) axiom links. Beffara [5] has shown a realisability semantics of Linear Logic which uses a synchronous version of the π-calculus with a dynamic link. The present work differs from these works (and also from [8,14,12]) in that we focus on significant, and independently conceived, fragments of the original formalisms, and that there is an exact correspondence in term/graph constructions and dynamics.

In the remainder, we first outline a fragment of proof-nets we shall use (which is a non-deterministic extension of polarised proof-nets in [27]). Then we outline a fragment of the π-calculus we shall use (which is a relaxing of type constraints in the typed π-calculus in [24]). We then relate these two formalisms. In the next section we discuss how incremental restrictions in these two formalisms lead to more restricted classes of processes/proofs. This is the core of our correspondence where we finally reach a precise coincidence between polarised proof-nets [27] and the control π-calculus [24]. The final section is dedicated to extensions of the correspondence between the two formalisms.

2. Proof-nets

The notion of proof-nets we consider here comes from polarised proof-nets [27,28] with two important extensions: n-ary !-nodes (responsible for non-determinism) and named cuts. In addition, it is presented in the so-called "focalised" form [17] (using n-ary multiplicative nodes, $\otimes$ and $\parr$); we omit axiom nodes; and we only consider exponential cuts.

The n-ary !-nodes correspond to a particular use of co-contraction nodes as defined in differential interaction nets [13]. An n-ary !-node can be implemented as n usual !-nodes followed by a n-ary co-contraction node (or $n-1$ binary co-contraction nodes).

Formulae. We consider positive and negative formulae given by the following grammar. Let $n \geq 0$ below.

$$\mathsf{P} ::= \bigotimes_{1\leq i\leq n} !\mathsf{N}_i$$
$$\mathsf{N} ::= \parr_{1\leq i\leq n} ?\mathsf{P}_i$$

The case of a 0-ary $\otimes$ (resp. $\parr$) is usually denoted by 1 (resp. $\bot$).

A formula is *positive* if it is of the form P, *negative* if it is in the form N.

The orthogonal of a formula, written $\mathsf{P}^\bot$ or $\mathsf{N}^\bot$, is obtained by exchanging $\otimes$ and $\parr$, ! and ?. Note taking the orthogonal of the orthogonal gives us the original formula.

Proof-nets. The following are proof-nets for polarised linear logic except we add slightly stronger constraints in the shape of formulae based on n-ary tensors/pars (i.e. we use the focalised form [17]), using, in addition to the formulae given above, those of the forms $!\mathsf{N}$ (which is different from a unary tensor $\bigotimes !\mathsf{N}$) and $?\mathsf{P}$ (which is different from a unary par). Moreover !-nodes are extended to the n-ary ($n > 0$) case.

We use the following four kinds of nodes: $\otimes$-nodes, $\parr$-nodes, ?-nodes, !-nodes given by:

$$\frac{!\mathsf{N}_1 \quad \cdots \quad !\mathsf{N}_n}{\bigotimes_{1\leq i\leq n} !\mathsf{N}_i} n \geq 0 \qquad \frac{?\mathsf{P}_1 \quad \cdots \quad ?\mathsf{P}_n}{\parr_{1\leq i\leq n} ?\mathsf{P}_i} n \geq 0 \qquad \frac{\mathsf{P} \quad \cdots \quad \mathsf{P}}{?\mathsf{P}} n \geq 0 \text{ premises} \qquad \frac{\mathsf{N} \quad \cdots \quad \mathsf{N}}{!\mathsf{N}} n > 0 \text{ premises}$$

with the corresponding formulae labelling edges.

Each node given above has the shape of a deduction rule: thus we use the standard terminology such as premise and conclusion of a node. Note we demand identical formulae as the premises of the third (resp. fourth) rule.

We construct a graph from these nodes together with labelled directed edges so that: (1) there is at most one outgoing edge from each node (conceived as going out from the conclusion of the source node); (2) if there is an edge from one node a to another node b, then the conclusion of a should occur as one of the premises of b (an edge is always directed from the conclusion of a node to a premise of another); and (3) if a node a has n premises, then there are always n incoming edges to a, which we label $1, 2, \ldots, n$, usually left implicit, such that the conclusion of the source node is covered by the ith premise of the source node through the ith edge, i.e. the premise of the target should be identical with the conclusion of the source as a formula (thus a premise of the target is justified by exactly one conclusion of the source). However we identify proof-nets up to permutation of the incoming edges of each of the ?-nodes and !-nodes.

By (3), each edge is associated with a formula, occurring in its source as the conclusion and in its target as one of the premises (the ith premise if the edge is labelled i). An edge is called *positive* (resp. *negative*) if the associated formula is positive or is of the form $!\mathsf{N}$ (resp. negative or is of the form $?\mathsf{P}$).

Note there can be no circular sequence of edges (since formulae are inductively given). If there is an edge from a node a to a node b, we say a is *above* b, or symmetrically b is *below* a.

A node without outgoing edge is called a *conclusion* of the whole proof-net.

A *proof-net* (without cuts) is a directed (and possibly disconnected) graph of this kind satisfying:

- Each ?-node has *zero or more* $\otimes$-nodes of the same conclusion above it. Each !-node has *one or more* $\parr$-nodes of the same conclusion above it.
- Each $\otimes$-node has zero or more !-nodes above it. Dually each $\parr$-node has zero or more ?-nodes above it.
- With each premise a of each !-node, we associate a *box*, that is a sub-proof-net with a conclusion a (called *the main conclusion of the box*), as well as zero or more $\otimes$-conclusions (called *the auxiliary conclusions of the box*), each of which must be above a ?-node (there are no other conclusions).
- Two boxes never overlap (they can be nested if they are associated with different !-nodes).

A node is *at depth n* if there are n boxes enclosing it. Note a node which does not occur in any box is at depth 0.

Cuts. We define *cuts* in a proof-net in a not completely usual, but equivalent, way following the idea in [17] that a cut corresponds to two *dual* edges (i.e. with orthogonal types) living at the same address (i.e. having the same name).

A *cut* is a pair of *dual* conclusions that occur in the same (zero or more) boxes (so in particular they are at the same depth), with one being the conclusion of a ?-node which is not a premise of another node and another being the conclusion of a !-node (which may or may not be a premise of another node).

A proof-net with cuts is given by a proof-net with zero or more cuts such that these cuts satisfy the *disjointness condition*, in the sense that two distinct cuts never share a node. This disjointness condition does not preclude non-deterministic dynamics as we shall discuss when we introduce cut eliminations.

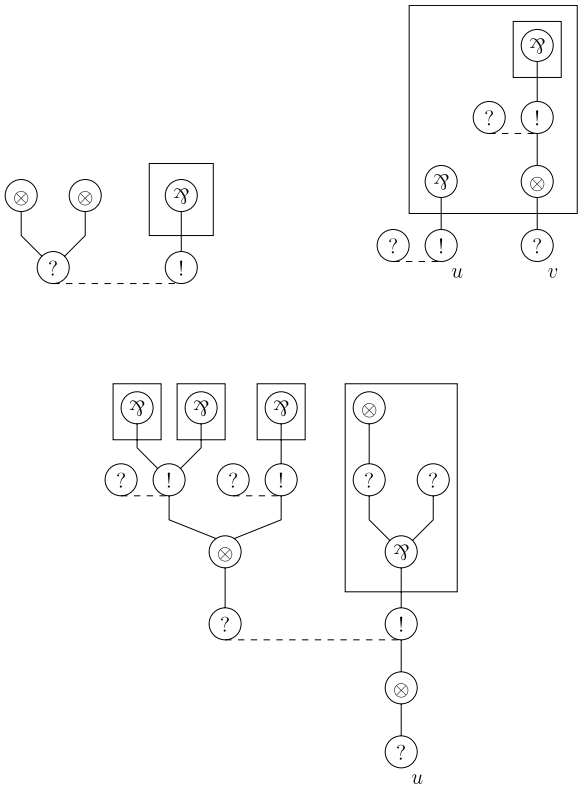


Fig. 1. Examples of proof-nets.

Example 1. Three examples of proof-nets are given in Fig. 1, where dashed lines represent cuts. The top-left proof-net has one box with the unique $\parr$-node above the !-node. It also has two $\otimes$-nodes sharing the ?-conclusion. The top-right proof-net has two (nested) boxes, whose outermost box has one auxiliary conclusion on the right and its main conclusion on the left. The conclusion of each !-node is cut with a ?-node without premises (the conclusion formula is implicit by duality). Finally the proof-net at the bottom has four boxes. The left-most two boxes share the same conclusion. None of these boxes contain auxiliary conclusions. In Section 4 later (Example 4), we shall present how these proof-nets can be translated to the equivalent π-terms.

Identification of nodes into a single node. We use the following operation on proof-nets. Consider two ?-nodes, say w_1 and w_2, with the same conclusion and without outgoing edges (i.e. their conclusions are not used for justifying the premises). Then the *identification* of w_1 and w_2 is done by replacing w_1 and w_2 by a single ?-node, say w, in such a way that each premise of w_1 and w_2 is now a premise of w. w_1 and w_2 disappear and their conclusions are now replaced by one (we do not have to worry about nodes below the original conclusions since, by our assumption, they are not used as premises of other nodes). The identification of two !-nodes in a proof-net is given in the same way.

Cut elimination. We consider a cut between a ?-node say w, having a $\otimes$-node above it, and a !-node say o, all of them being at depth 0. Cut elimination eliminates this cut, and possibly generates new nodes. This corresponds to the τ-action (reduction) in the π-calculus. This is done as follows.

First, recall a !-node has *one or more* $\parr$-nodes above it, hence the corresponding boxes. We choose any one of these $\parr$-nodes, say p, and the box b above p, and make a single copy of its content. When copying, if an auxiliary conclusion (i.e. a $\otimes$-conclusion) of the box justifies a premise of a ?-node through an edge, then we add the corresponding premise to the ?-node, so that there is an edge from the conclusion of the copied $\otimes$-node to this newly added premise (note this increases the number of premises of this ?-node). Dually we choose one of the $\otimes$-nodes at depth 0 above w, say t.

Second, we remove the selected $\parr$-node (which was originally p) from the copy of b. This leaves $k \geq 0$?-conclusions in the copy. Dually we erase the $\otimes$-node t, leaving k !-nodes. Note their numbers are equal because a cut can only be placed between dual formulae.

Third and finally, we put a cut between the conclusion of each of these k ?-nodes and the corresponding !-node one by one, following their labels. However if two such associated nodes o' (the !-node) and w' (the ?-node) are such that o' has already been cut with a ?-node w'', we perform the *identification* of w' and w'' in the sense defined above, instead of adding a new cut (which would have violated the disjointness condition for cuts, cf. page 5).

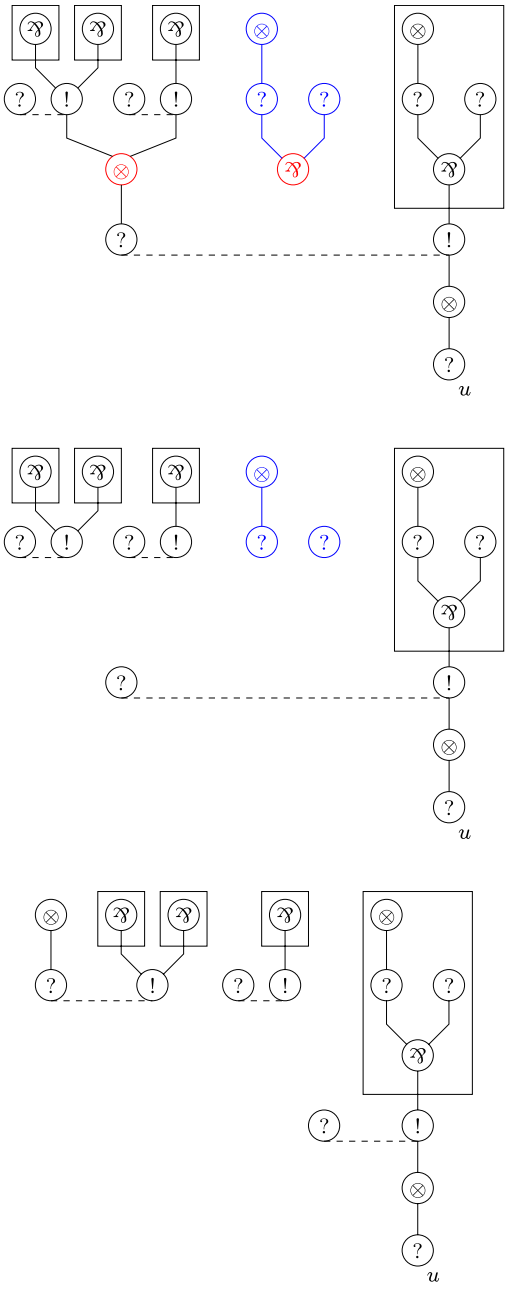


Fig. 2. Cut elimination procedure.

Anticipating the correspondence result in Section 4, here is an informal illustration of how these three steps of cut elimination above are related to ideas in the π-calculus. The first step above corresponds to a non-deterministic selection of one of the redexes in the π-term, by choosing an input ($\parr$-node above the !-node) and an output ($\otimes$-node above the ?-node), followed by making a copy of the body of the replicated input in that redex. The second and third steps graphically carry out the standard (internal) name passing interaction (a similar decomposition is discussed in [34]). In the second step, the initial part of name passing is performed, annihilating a pair of the initial subjects of the prefixed process (a *subject* is the initial name occurrence in a prefix, e.g. the initial x in $x(y).P$). The third step then carries out the rest of the interaction, i.e. value passing, which is bound (private) name passing in the present context. The names emitted and the names received are simultaneously identified and hidden, forming new pairs of potential redexes, signified by new cuts.

Example 2. Using the proof-net at the bottom in Fig. 1, the three steps of cut elimination are shown in Fig. 2. The first picture is after the first step, copying the content of the right-most box. Choosing a $\parr$-node and a $\otimes$-node is trivial (since each is unique). Further, because there is no auxiliary conclusion, we need no additional edges to newly made premises. In the

second picture, we simply eliminate the pair of $\otimes$- and $\parr$-nodes. The third picture gives the final result, by identifying the two ?-nodes on the left with their counterparts on the copied graph (since the conclusions of the ?-nodes have already been cut).

In addition to this standard cut elimination, we also consider an extended version of cut elimination, which adds the following two cases. First, we consider the case when a premise ($\otimes$-node) of the ?-node w is *inside a larger box* b' (so that it is not of depth 0) but not in the selected box b of the !-node o, in which case the copy of the content of b is moved inside this enclosing box b' and we proceed as for the standard case. Second, we reduce a cut between a 0-ary ?-node and a conclusion !-node (i.e. not above another node) by erasing the !-node and its boxes and the associated premises of ?-nodes. We call *extended cut elimination* the result of adding these last two cases.

Complete proof-nets. Since proof-nets have a slightly finer presentation of constructions than the π-calculus, we restrict the shape of proof-nets by a completeness constraint, as well as adding names as labels on conclusions of proof-nets (and on no other node) to get an explicit matching with channel names.

The following conditions are required for *complete proof-nets*:

- Each $\otimes$-node is above a ?-node. Each $\parr$-node is above a !-node.
- The conclusion of each !-node must be cut with the conclusion of a ?-node.
- Some conclusions of the proof-net are labelled with names according to the following rules:
  - If its node is a ?-node then it must have a label if and only if it does not belong to a cut.
  - If its node is a !-node then it must have a label if it does not belong to a cut. If it does, it may or may not have a label.

  Two distinct conclusion nodes are never labelled with the same name. We write $x, y, \ldots$ for names used to label conclusions.

If a cut contains the conclusion of a !-node which is either labelled by a name or a premise of another node, the cut is called *named* (and *unnamed* otherwise). We restrict the second case of extended reduction, which we may call *erasing reduction*, to unnamed cuts (because a named cut may add more $\otimes/\parr$-nodes above ?/!-nodes, making non-trivial cut elimination possible: the erasing reduction is intended to eliminate the cut which can never induce interaction).

A node is *free* if its conclusion is named (hence unconnected to any premise). In this case we also say this conclusion is free. In a complete proof-net, we will only consider free conclusions as the conclusions of the proof-net. In particular a !- or ?-node which is not a premise of any node and which has an unnamed conclusion (so that it must belong to a cut) is now allowed to occur inside a box, which we do not consider as an auxiliary conclusion of the box. This extends the previous definition of box without cut (page 5), but we stick to the fact that auxiliary conclusions of a box are conclusions of $\otimes$-nodes.

There is a simple completion procedure turning any proof-net into a complete one. Let $\mathcal{R}$ be a proof-net without names labelling the conclusions (otherwise we can just remove them), first we complete $\mathcal{R}$ so that it has neither free $\otimes$-nodes nor free $\parr$-nodes, as follows: if there is a free $\otimes$-node with conclusion P then add a new ?-node just below it which is from P to ?P. If there is a free $\parr$-node, simply erase this $\parr$-node. For the conclusion of each !-node which does not belong to a cut, introduce a new ?-node (with 0 premise) and add a cut between the conclusion of the !-node and the conclusion of this new node. Finally, we add pairwise distinct names to all the conclusions of the resulting proof-net which are not connected to premises, except that we do not name the conclusions of ?-nodes belonging to cuts.

Example 3. Each proof-net in Fig. 1 is complete.

Computational intuition behind completion. For those familiar with polarised proof-nets, the completion is a simple procedure to turn visible interface of proof-nets to specific shape (! and ?).

For readers from the process algebra background, we supply some computational intuition. First, a $\otimes$-node (resp. a $\parr$-node) indicates abstraction of names for input (resp. ν-abstraction for bound output), but *without the initial subject*, so that such a node with n premises corresponds to a n-adic communication action — receiving or sending n names. Thus one may consider them as indicating the $(y_1..y_n)$-part of $!x(y_1..y_n).P$ and $\bar{x}(y_1..y_n)P$ for $\parr$ and $\otimes$, respectively. A ?-node (resp. !-node) represents a channel used for the subject of prefixes (x in $!x(y_1..y_n).P$ and $\bar{x}(y_1..y_n)P$) and, simultaneously, the sharing of this channel by the prefixes: that is, if a ?-node has m premises, this corresponds to m output processes sharing the same channel (if $m = 0$, then this means there is no process sharing that channel). Thus the completion procedure starts from a graph representing, say, the union of $(y_1..y_n).P$ and $(z_1..z_n)Q$ and reaches the process $P|\bar{w}(z_1..z_n)Q$. A box denotes a synchronisation boundary, i.e. $(y_1..y_n).P$ of $!x(y_1..y_n).P$ is in the box (note a box exists only for !, so the correspondence is exact). Finally cuts are when two processes can potentially interact by one having an input and another an output on a shared channel. Unnamed cuts in addition add hiding of that shared channel.

3. Typed π-calculus

We consider a polyadic π-calculus with the following restrictions: communication is private (or internal) [35,38] and asynchronous [21,9] and moreover all and only inputs are replicated. The typing system will enforce an additional locality constraint (channels received by an input are only used for output). From a different viewpoint this calculus is a straightforward non-deterministic and non-terminating extension of the calculus in [24] used for embedding $\lambda\mu$-calculus fully abstractly.

Terms. Given a set of names denoted $x, y, \ldots$, terms are given by:

$$P ::= !x(\vec{y}).P \quad | \quad \bar{x}(\vec{y})P \quad | \quad P|Q \quad | \quad \nu xP \quad | \quad 0.$$

$!x(\vec{y}).P$ is called a replicated receptor, or input; $\bar{x}(\vec{y})P$ is an output; $P|Q$ is a parallel composition; νxP is hiding of x; 0 is the inaction.

In input and output constructions, the $\vec{y}$ in $!x(\vec{y}).P$ and $\bar{x}(\vec{y})P$ are bound as for x in νxP.

We may call the output construction $\bar{x}(\vec{y})P$ we use, which corresponds to $\overrightarrow{\nu y}(\bar{x}\langle\vec{y}\rangle|P)$ in the usual π-calculus, *private output*.

Types and judgements. Terms are typed with input/output types given as follows.

$$\begin{array}{lll} \tau_{\mathtt{I}} & ::= & (\vec{\tau_0})^{!} \\ \tau_0 & ::= & (\vec{\tau_{\mathtt{I}}})^{?} \\ \tau & ::= & \tau_{\mathtt{I}} \quad | \quad \tau_0 \end{array}$$

The *mode* of a type is defined by $\mathsf{md}(\tau_{\mathtt{I}}) = \mathtt{I}$ and $\mathsf{md}(\tau_0) = \mathtt{O}$.

The *dual* $\overline{\tau}$ of a type τ is obtained by exchanging $\tau_{\mathtt{I}}$ and τ_0, ! and ?.

Intuitively, a name can have both a type and its dual type, since a single name is shared by both inputs and outputs. In typing judgements, we use the output form of the type if we are sure that no input occurs on that name, otherwise using the input form of the type.

A *context* A is a finite function from names to types. If x is not in the domain of A, we write $A, x : \tau$ for the context which extends A with $x \mapsto \tau$ (otherwise $A, x : \tau$ is not defined). According to the previous remark, when adding a declaration to a context we use the following partial operation $\odot$:

$$\begin{array}{ll} A \odot x : \tau = A, x : \tau & \text{if } x \notin A \\ A \odot x : \tau \text{ is not defined} & \text{if } x : \tau' \in A \text{ with } \tau' \neq \tau \text{ and } \tau' \neq \overline{\tau} \\ A \odot x : \tau_0 = A & \text{if } x : \tau_0 \in A \text{ or } x : \overline{\tau_0} \in A \\ A \odot x : \tau_{\mathtt{I}} = A', x : \tau_{\mathtt{I}} & \text{if } x : \overline{\tau_{\mathtt{I}}} \in A \text{ and } A' = A \setminus x \\ A \odot x : \tau_{\mathtt{I}} = A & \text{if } x : \tau_{\mathtt{I}} \in A \end{array}$$

We extend $\odot$ to a partial operation on contexts as follows: given $A_1 = x_1 : \tau_1, \ldots, x_n : \tau_n$ and $A_2 = y_1 : \tau'_1, \ldots, y_k : \tau'_k$, $A_1 \odot A_2$ is given as $A_1 \odot A_2 = ((A_1 \odot y_1 : \tau'_1) \cdots) \odot y_k : \tau'_k$. $A_1 \odot A_2$ is defined iff all of the involved $\odot$-compositions are defined, with the same resulting value: otherwise it is undefined.

Lemma 1 (*Operator* $\odot$). *Define the partial order* $\leq$ *on contexts generated from the set inclusion and* $A, x : \tau_0 \leq A, x : \overline{\tau_0}$ *for each* τ_0 *(cf. [20]). Then if* $A_1 \odot A_2$ *is defined then it gives the join of* A_1 *and* A_2 *with respect to* $\leq$.

Proof. Immediate from the definition. □

Intuitively, the ordering $\leq$ represents the degree of composability: the smaller a type is in this ordering, the easier that type is composable with others (i.e. $\odot$ is more defined). By Lemma 1, $\odot$ defines the (partial) join under the ordering $\leq$, hence as an operation $\odot$ is partially commutative and partially associative (i.e. definedness and, if defined, the resulting values coincide between $A \odot B$ and $B \odot A$, similarly $(A \odot B) \odot C$ and $A \odot (B \odot C)$), with the identity the empty context. In particular, assuming x is not in $\mathsf{dom}(A_1) \cup \mathsf{dom}(A_2)$, $A_1 \odot (A_2, x : \tau)$ and $(A_1 \odot A_2), x : \tau$ coincide in their definedness and their value.

A typing judgement is of the shape $\vdash P \triangleright A$. We present the typing rules.

Typing rules. The following typing rules relax some of the constraints of those from [6]. Below $\emptyset$ is the empty context.

$$\frac{}{\vdash 0 \triangleright \emptyset}\,0 \qquad \frac{\vdash P_1 \triangleright A_1 \qquad \vdash P_2 \triangleright A_2}{\vdash P_1|P_2 \triangleright A_1 \odot A_2}\,| \qquad \frac{\vdash P \triangleright A, x : \tau_{\mathtt{I}}}{\vdash \nu xP \triangleright A}\,\nu$$

$$\frac{\vdash P \triangleright A, \vec{y} : \vec{\tau_0} \qquad \mathsf{md}(A) = \mathtt{O}}{\vdash !x(\vec{y}).P \triangleright A \odot x : (\vec{\tau_0})^{!}}\,\mathrm{in} \qquad \frac{\vdash P \triangleright A, \vec{y} : \vec{\tau_{\mathtt{I}}}}{\vdash \bar{x}(\vec{y})P \triangleright A \odot x : (\vec{\tau_{\mathtt{I}}})^{?}}\,\mathrm{out}$$

$$\frac{\vdash P \triangleright A}{\vdash P \triangleright A, x : \tau_0}\,\mathrm{wk}$$

where $\mathsf{md}(A) = \mathtt{O}$ means that all the types appearing in A are of mode $\mathtt{O}$. In each of (in), (out) and (|), we stipulate that the deduction is possible only when $\odot$ in the conclusion is defined. In (wk), we assume x does not occur in A. Below and henceforth we always assume the standard bound name convention, i.e. binding names are disjoint from free names.

Lemma 2 (*Permutation of wk*). *The application of* (wk) *can always be permuted up (hence with the same height of the derivation tree) except when the previous rule is* (0).

Proof. By rule induction and because, by Lemma 1, a disjoint union commutes with other composition by $\odot$. □

The typing is closed under injective renaming as is immediate from the shape of each typing rule (formally by a completely trivial rule induction on typing rules). We shall also use later the following result.

Lemma 3. *If* $\vdash P \triangleright A, x : \tau_0, y : \tau_0$ *then* $\vdash P\{x/y\} \triangleright A, x : \tau_0$. *If* $\vdash P \triangleright A, x : \tau_{\mathrm{I}}, y : \overline{\tau_{\mathrm{I}}}$ *then* $\vdash P\{x/y\} \triangleright A, x : \tau_{\mathrm{I}}$.

Proof. Both statements are proved by induction on typing rules. For the first statement the only non-trivial case is when the newly added subject of the output prefix (the subject is the initial name of a prefix) in (out) is coalesced, which is immediate by Lemma 1 which says that τ_0 is idempotent with respect to $\odot$. For the second statement the only non-trivial case is when we coalesce the newly added replicated name in (in), which is again direct from Lemma 1 noting the input type is bigger than its dual. □

The typing rules also satisfy the standard properties of weakening (by a disjoint context only including output types in its range) and thinning (in the sense that the part of a context whose domain is disjoint from free names of a process can be cut off — note this can only concern names with output types).

Structural congruence. The α*-equivalence* of terms is defined by the usual renaming of bound variables, avoiding unwanted capture of names.

The *structural congruence* relation on terms is the smallest congruence generated by the α-equivalence and the following equations:

$$0|P \equiv P \tag{1}$$
$$P|Q \equiv Q|P \tag{2}$$
$$P|(Q|R) \equiv (P|Q)|R \tag{3}$$
$$\nu x\nu y P \equiv \nu y\nu x P \tag{4}$$
$$\nu x(P|Q) \equiv (\nu x P)|Q \quad \text{if } x \notin Q \tag{5}$$
$$\bar{x}(\vec{u})\bar{y}(\vec{v})P \equiv \bar{y}(\vec{v})\bar{x}(\vec{u})P \quad \text{if } x \notin \vec{v} \text{ and } y \notin \vec{u} \text{ and } \vec{u} \cap \vec{v} = \emptyset \tag{6}$$
$$\bar{x}(\vec{u})(P|Q) \equiv (\bar{x}(\vec{u})P)|Q \quad \text{if } \vec{u} \notin Q \tag{7}$$
$$\nu y\bar{x}(\vec{u})P \equiv \bar{x}(\vec{u})\nu y P \quad \text{if } y \notin \{x, \vec{u}\} \tag{8}$$

Up to the equivalence relation $\equiv$, π-terms can be rewritten into *canonical forms*:

$$P \equiv \overrightarrow{\nu x}\overrightarrow{\bar{y}(\vec{z})}\left(\prod_{i\in I} !v_i(\vec{w}_i).P_i\right)$$

For the proof, first externalise all ν-bound channels which are not under input prefixes, then do the same for all output prefixes. Since there are only a finite number of output prefixes the remaining processes are input processes with no ν-binding.

The typing system is well behaved with respect to this structural congruence relation:

Lemma 4. *If* $\vdash P \triangleright A$ *and* $P \equiv Q$ *then* $\vdash Q \triangleright A$.

Proof. By Lemma 2 we safely assume, except when the process is 0, the last rule follows the uppermost constructor of the process. The initial equation (identity of 0 for $|$) is by weakening. The next two (commutativity and associativity of $|$) is by partial commutativity and associativity of $\odot$ from Lemma 1. νxyP and νyxP have the same contexts since ν just takes off a singleton from a context. The next rule, scope extrusion, is immediate from idempotence of an output type (again by Lemma 1). The next rule is again by commutativity of $\odot$ by Lemma 1. The second rule to the last, the second scope extrusion rule, is as the first scope extrusion rule. Finally the last rule is immediate from inspection of the two involved rules. □

Reduction. The dynamics of the calculus is defined by a reduction relation between terms.

$$\frac{}{!x(\vec{y}).P|\bar{x}(\vec{y})Q \to !x(\vec{y}).P|\nu\vec{y}(P|Q)} \qquad \frac{P \to P'}{P|Q \to P'|Q} \qquad \frac{P \to Q}{\nu x P \to \nu x Q} \qquad \frac{P \to Q}{\bar{x}(\vec{y})P \to \bar{x}(\vec{y})Q}$$
$$\frac{P \equiv P' \quad P' \to Q' \quad Q' \equiv Q}{P \to Q}$$

The *extended reduction* is obtained by replacing the first reduction rule above with the following two:

$$!x(\vec{y}).P|C[\bar{x}(\vec{y})Q] \searrow !x(\vec{y}).P|C[\nu\vec{y}(P|Q)]$$
$$\nu x!x(\vec{y}).P \searrow 0$$

where $C[\ ]$ is an arbitrary context not binding x. Note $\to \subset \searrow$. We have:

Lemma 5. *If* $\vdash P \triangleright A$ *and* $P \searrow Q$ *then* $\vdash Q \triangleright A$.

Proof. For the first rule we show that $\bar{x}(\vec{y})Q'$ and $\nu\vec{y}(P'|Q')$ have the same typing which is immediate by noting the binding by a private output and that by ν both demand the name to be typed by an input type and that an output type is idempotent and can be weakened. For the second rule we show that $\nu x!x(\vec{y}).P'$ and 0 have the same typing. The context for P' only contains the output types hence $\nu x!x(\vec{y}).P'$ can only contain them which can be weakened for 0. □

4. Translations

There is an immediate correspondence between negative formulae and output types and between positive formulae and input types. We turn it into a correspondence between our typing derivations for the π-calculus and the complete proof-nets.

From π-terms to proof-nets. A typing derivation with conclusion $\vdash P \triangleright A$ is translated as a complete proof-net $\mathcal{R}$ with labelled conclusions corresponding to A in such a way that: if $x : \tau_0 \in A$ (resp. $x : \tau_{\mathrm{I}} \in A$), $\mathcal{R}$ has a conclusion labelled by x with type the negative (resp. positive) formula corresponding to τ_0 (resp. τ_{I}). Moreover any positive conclusion belongs to a cut (following the definition of a complete proof-net).

Derivations are translated by:

- For $!x(\vec{y}).P$, we start with, by induction, a proof-net representing P, which has only ?-conclusions. We put a $\parr$-node between the conclusions labelled $\vec{y}$, we remove these labels, and we add a unary !-node under it with conclusion labelled x: the box associated with the premise of the !-node is the whole proof-net except the !-node and the free ?-nodes. Finally we cut the !-node with the ?-node (if any) whose conclusion is labelled x (and we remove the label of the ?-node), or with an augmented 0-ary ?-node if it does not exist.
- For $\bar{x}(\vec{y})P$, by induction we already have a net corresponding to P. We add a $\otimes$-node between the conclusions labelled $\vec{y}$, and we remove these labels. If there is already a negative edge with name x, we add the conclusion of the $\otimes$-node as a new premise to the corresponding ?-node. If not, we introduce a new unary ?-node with conclusion labelled x below the new $\otimes$-node.
- $P_1|P_2$ is translated as the juxtaposition of the two graphs with some identifications of free nodes (identification of nodes is defined in Section 2, page 7):
  - if x has input type in both P_1 and P_2, we identify the two ?-nodes corresponding to it, and the two !-nodes (and thus the two named cuts become one by set union);
  - if x has output type in both P_1 and P_2, we identify the two ?-nodes corresponding to it;
  - if x has output type in P_1 and input type in P_2 (or the converse), we identify the ?-node corresponding to the output occurrence and the ?-node which is cut with the !-node corresponding to the input occurrence. The label x on the conclusion of the ?-node is removed.
- To translate νxP, we first translate P. We then erase the name x on the conclusion of the !-node with conclusion type τ_{I} (the named cut containing this edge becomes an unnamed one).
- 0 is translated as the empty graph.
- The weakening rule is translated by the introduction of a 0-ary ?-node with conclusion labelled x.

From proof-nets to π-terms. Given a complete proof-net $\mathcal{R}$, we build a typing derivation for a judgement $\vdash P \triangleright A$ where A contains typing declarations for the labels of the conclusions of $\mathcal{R}$ (with the input type if a positive conclusion has label x and the output type otherwise).

The following construction of the translation is by induction on the size of $\mathcal{R}$ (the size of a proof-net is $n + 2k + p$ where n is the number of nodes, k is the number of $\otimes$-nodes which are premise of a ?-node involved in a cut, named or not, and p is the number of unnamed cuts).

- If $\mathcal{R}$ is empty, it corresponds to $\vdash 0 \triangleright \emptyset$.
- If $\mathcal{R}$ has a non-0-ary ?-node at depth 0 named x, it has one or more, say m, $\otimes$-nodes above it. We take off each of these m $\otimes$-nodes, then we get a net with $m \times n$ free !-nodes above these $\otimes$-nodes, for the arity n of $\otimes$ (same arity for all these $\otimes$-nodes by typing). By induction we can translate this net into $\vdash P \triangleright A, \vec{y}_1 : \vec{\tau_{\mathrm{I}}}, \ldots, \vec{y}_m : \vec{\tau_{\mathrm{I}}}$. Then $\vdash \bar{x}(\vec{y}_1)..\bar{x}(\vec{y}_m)P \triangleright A, x : (\vec{\tau_{\mathrm{I}}})^?$ (where length of $\vec{y}_j$ is n, $1 \leq j \leq m$) is the required translation.
- If $\mathcal{R}$ has a 0-ary final ?-node, we remove it, we apply the induction hypothesis and we apply a weakening rule.
- If $\mathcal{R}$ contains an unnamed cut at depth 0, we turn it into a named one by adding a new name x for its positive conclusion. By induction hypothesis, we have a typing derivation ending with $\vdash P \triangleright A, x : \tau_{\mathrm{I}}$ and we apply the (ν) rule to it.
- If $\mathcal{R}$ contains a named cut at depth 0 with name x and involving the conclusion of a non-0-ary ?-node, take off that cut, then name the ?-node y and !-node x with y fresh, add a new named cut between the !-node and an augmented 0-ary ?-node, then translate the result by induction. We get $\vdash P \triangleright A, x : \tau_{\mathrm{I}}, y : \overline{\tau_{\mathrm{I}}}$, and take $\vdash P\{x/y\} \triangleright A, x : \tau_{\mathrm{I}}$ by Lemma 3.
- If $\mathcal{R}$ contains a unique outermost box whose main conclusion (which is a premise of a !-node by definition) is named x, and if $\mathcal{R}$ contains no other nodes except the ?-nodes under the $\otimes$-conclusions of the box and the 0-ary ?-node cut with the !-node, then we remove the !-node (with the ?-node cut with it) and the $\parr$-node above with the introduction of new names $y_1, \ldots, y_n$ as labels for its premises which are ?-nodes. By induction hypothesis, we get a derivation with conclusion $\vdash P \triangleright A, \vec{y} : \vec{\tau_0}$ (notice that in this case $\mathrm{md}(A) = 0$) and we derive $\vdash !x(\vec{y}).P \triangleright A \odot x : (\vec{\tau_0})^!$.

- If none of the previous cases applies, $\mathcal{R}$ contains more than one box at depth 0, i.e. more than one outermost box. If it contains n outermost boxes, we split it into n proof-nets: one for each box (containing the box, an associated !-node still with the same name but with a single $\parr$-node above, and a 0-ary ?-node cut with it) with a splitting of the ?-nodes. By induction hypothesis, we have $\vdash P_1 \triangleright A_1, \ldots, \vdash P_n \triangleright A_n$ and we get $\vdash ((P_1|P_2)|\ldots)|P_n \triangleright A_1 \odot A_2 \odot \cdots \odot A_n$. The $\odot$ operation taking exactly into account the identifications of names.

Above we assume we can choose a ?-node, a !-node etc. used in each case uniquely by e.g. a certain ordering on nodes. The choice of this ordering does not change the resulting processes, as we shall see soon.

Example 4. According to the translation given above, the following three π-terms correspond to the proof-nets of Fig. 1:

$$\nu x(\bar{x} \mid \bar{x} \mid !x.0)$$
$$!u.\bar{v}(x)!x.0$$
$$\bar{u}(x)\left(\bar{x}(y'z')(!y'.0 \mid !y'.0 \mid !z'.0) \mid !x(yz).\bar{y}\right)$$

where we write, following the standard convention [35], $!x.P$ for $!x().P$ and $\bar{x}$ for $\bar{x}()0$. In the first term, we have the shared redex at a channel named x, with two messages at its output end and one input at its input end (the two messages correspond to two $\otimes$-nodes above the ?-node). For the second term, x corresponds to the auxiliary $\otimes$-conclusion of the box, while the free v corresponds to the similarly named ?-node. For the third term, the cut elimination depicted in Fig. 2 leads to, in one step, the following term:

$$\bar{u}(x)((\nu y'z')\left(!y'.0 \mid !y'.0 \mid !z'.0 \mid \bar{y'}\right) \mid !x(yz).\bar{y})$$

This term corresponds to the proof-net at the bottom in Fig. 2. The three steps in Fig. 2 roughly correspond to the choice of a redex at x (unique in this case), copying the body $\bar{y}$ of the left-most replication, and identifying the first and second output parameters $y'z'$ and the first and second input parameters yz (in the copy) respectively.

Structural congruence and σ-equivalence. We confirm several basic properties of the translations. First, two typable π-terms $\vdash P \triangleright A$ and $\vdash Q \triangleright A$ are called *σ-equivalent* (in relation with the corresponding notion for the λ-calculus [37]), denoted $\vdash P \simeq_\sigma Q \triangleright A$, if they translate into the same proof-net.

Proposition 1. *Let $\vdash P \triangleright A$ and $\vdash Q \triangleright A$ be two typed π-terms,*

$$P \equiv Q \iff \vdash P \simeq_\sigma Q \triangleright A.$$

Proof. The implication $P \equiv Q \Rightarrow \vdash P \simeq_\sigma Q \triangleright A$ is trivial except the following cases:

- $\bar{x}(\vec{u})\bar{y}(\vec{v})P \equiv \bar{y}(\vec{v})\bar{x}(\vec{u})P$ (with $x \notin \vec{v}$ and $y \notin \vec{u}$ and $\vec{u} \cap \vec{v} = \emptyset$): if x and y are different, the result is immediate, otherwise both $\bar{x}(\vec{u})$ and $\bar{x}(\vec{v})$ add a $\otimes$-node above the ?-node corresponding to x and the order does not matter.
- $\bar{x}(\vec{u})(P|Q) \equiv (\bar{x}(\vec{u})P)|Q$ (with $\vec{u} \notin Q$): the only interesting case is when $x \in Q$, in which case we just have to remark that in the first term a $\otimes$-node is added to the common ?-node coming from P and Q and corresponding to x, while in the second term the $\otimes$-node is added to a ?-node coming from P and then it is merged with the ?-node with name x coming from Q.
- $\nu y\bar{x}(\vec{u})P \equiv \bar{x}(\vec{u})\nu yP$ (with $y \notin \{x, \vec{u}\}$): the νy operation hides the conclusion of a cut node which is not concerned by the $\bar{x}(\vec{u})$ operation.

In the other direction: $\vdash P \simeq_\sigma Q \triangleright A \Rightarrow P \equiv Q$, we can consider the particular case where P and Q are canonical forms (otherwise $P \equiv P_0$ and $Q \equiv Q_0$ with P_0 and Q_0 canonical forms entails $P_0 \simeq_\sigma P \simeq_\sigma Q \simeq_\sigma Q_0$ and by this particular case $P \equiv P_0 \equiv Q_0 \equiv Q$).

Let $\mathcal{R}$ be the translation of P and Q. We work by induction on $\mathcal{R}$. If $\mathcal{R}$ contains unnamed cuts at depth 0, we introduce conclusions with fresh names for these cuts. This exactly corresponds to removing the ν constructions in head position in P and Q, and by typability we cannot have νs on non-free variables. Using $\equiv$, the order of the νs does not matter and we can apply the induction hypothesis.

Observe that two output prefixes $\bar{x}_1(\vec{u}_1)$ and $\bar{x}_2(\vec{u}_2)$ commute by $\equiv$ as soon as the conclusion of the corresponding $\otimes$-nodes goes to named ?-nodes (more generally, if one is not above the other in $\mathcal{R}$). We consider a $\otimes$-node at depth 0 above a named ?-node. By the remark, P and Q are equivalent to terms starting with the corresponding $\bar{x}(\vec{u})$. We remove it and we apply the induction hypothesis.

We now consider the case where $\mathcal{R}$ does not contain any $\otimes$-node at depth 0. We split $\mathcal{R}$ according to its boxes. This corresponds to splitting both P and Q through parallel composition (which is commutative and associative according to $\equiv$).

Finally, if $\mathcal{R}$ is reduced to one box (with associated !, ? and cut), we have $P = !x(\vec{y}).P'$ and $Q = !x(\vec{y}).Q'$ and we apply the induction hypothesis to P' and Q'. □

Second we show that the two translations are mutually inverse.

Proposition 2. *If we translate $\mathcal{R}$ into a typed process, and this typed process into a proof-net $\mathcal{R}'$ back again, then $\mathcal{R}$ and $\mathcal{R}'$ are isomorphic. Symmetrically, if we translate $\vdash P \triangleright A$ into a proof-net, and this proof-net into a process say $\vdash Q \triangleright A$ back again, then $P \equiv Q$.*

Proof. In each direction, we use rule induction on the translation rules. All cases are mechanical. For example, if we translate $\mathcal{R}$ by choosing a non-0-ary ?-node, then this is translated into a sequence of outputs at the same subject, say x, in the form:

$$\bar{x}(\vec{y}_1)..\bar{x}(\vec{y}_n)P$$

where x does not occur in P. Apply the translation of this term and we obtain precisely the same net as before. Similarly the other direction. □

Corollary 1. *The translation of $\mathcal{R}$ as given before is determined uniquely up to $\equiv$ regardless of the choices of nodes in translation.*

Proof. Suppose two different ways to translate $\mathcal{R}$ result in $\equiv$-distinct processes. Then their translations do not result in the same net by Proposition 1, contradicting Proposition 2. □

Simulations. We turn our attention to dynamics.

Theorem 1. *$P \to Q$ if and only if the translation of $\vdash P \triangleright A$ reduces to the translation of $\vdash Q \triangleright A$ by cut elimination at depth 0. $P \searrow Q$ if and only if the translation of $\vdash P \triangleright A$ reduces to the translation of $\vdash Q \triangleright A$ by extended cut elimination at depth 0.*

Proof. If P reduces to P' (with $\vdash P \triangleright A$ and thus $\vdash P' \triangleright A$ by Lemma 5), then there is an active input and an active output sharing the same channel, say x. By structural equality (which is harmless by Proposition 1), we can transform the process so that all possible redexes at x can be juxtaposed:

$$\Pi_i !x(\vec{y}).P_i | \Pi_j \bar{x}(\vec{y})Q_j$$

(with an enclosing reduction context which does not contain an input at x). By translation this becomes a named cut (or an unnamed one if the enclosing context contains νx). The rest follows by observing the precise correspondence between the reduction in the π-calculus and the construction of a cut elimination. First, we choose one input and one output at x from the above, which corresponds to the choice of one $\parr$-node and one $\otimes$-node in the cut elimination. Then we make the copy and pass names to obtain, in the place of $\bar{x}(\vec{y})Q_j$:

$$(\nu\vec{y})(P_i|Q_j)$$

which corresponds to making the copy of the content of a box (corresponding to copying P_i above from under the replication), making new cuts at $\vec{y}$ (corresponding to |), and hiding these cuts (corresponding to $(\nu\vec{y})$). Note each step exactly corresponds to each other in both ways.

If $\mathcal{R}$ reduces to $\mathcal{R}'$, then there is an either named or unnamed cut. Translate this cut (which we can assume named without loss of generality, by considering a name hiding operation at the level of proof-nets), by which we obtain a term of the form $\Pi_i !x(\vec{y}).P_{1i}|\Pi_j \bar{x}(\vec{y})P_{2j}$, where Π denotes n-fold composition. The rest follows again by the correspondence between the construction of cut elimination and the reduction. In each direction, it is crucial that only the redex changes the shape: other parts remain the same, hence we can use induction. The cases for extended reduction are similar. □

5. Additional constraints

This section incrementally adds constraints on polarised proof-nets and processes one by one, leading to the coincidence of their fragments which fully abstractly embed the call-by-name and call-by-value $\lambda\mu$-calculi (or, equivalently, a simply typed λ-calculus extended with callcc).

5.1. Reception determinism

If we modify the definition of $\odot$ by demanding that $(A, x : \tau_{\mathtt{I}}) \odot x : \tau_{\mathtt{I}}$ is no longer defined (thus constraining the (|)-rule), we get proof-nets in which each !-node has exactly one premise. Both constraints act locally and in precise correspondence with each other, hence the same simulation result as Theorem 1 holds.

Computationally this means that all messages to the same input name (i.e. messages going through the same channel) always reach a *unique* replicated input process [39,6].

5.2. Emission determinism

We define, following [6], a partial operation $\boxdot$ on modes:

$$\mathtt{I} \boxdot \mathtt{I} = \mathtt{I}$$
$$\mathtt{I} \boxdot \mathtt{O} = \mathtt{O}$$
$$\mathtt{O} \boxdot \mathtt{I} = \mathtt{O}$$

and stipulate $\mathtt{O} \boxdot \mathtt{O}$ is not defined.

We add to judgements an input/output mode ϕ: $\vdash_\phi P \triangleright A$.

$$\frac{}{\vdash_{\mathtt{I}} 0 \triangleright \emptyset} \qquad \frac{\vdash_{\phi_1} P_1 \triangleright A_1 \qquad \vdash_{\phi_2} P_2 \triangleright A_2}{\vdash_{\phi_1 \boxdot \phi_2} P_1 | P_2 \triangleright A_1 \odot A_2} \qquad \frac{\vdash_\phi P \triangleright A, x : \tau_{\mathtt{I}}}{\vdash_\phi \nu x P \triangleright A}$$

$$\frac{\vdash_{\mathtt{0}} P \triangleright A, \vec{y} : \vec{\tau_{\mathtt{0}}} \qquad \mathsf{md}(A) = \mathtt{0}}{\vdash_{\mathtt{I}} !x(\vec{y}).P \triangleright A \odot x : (\vec{\tau_{\mathtt{0}}})^!} \qquad \frac{\vdash_{\mathtt{I}} P \triangleright A, \vec{y} : \vec{\tau_{\mathtt{I}}}}{\vdash_{\mathtt{0}} \bar{x}(\vec{y}) P \triangleright A \odot x : (\vec{\tau_{\mathtt{I}}})^?}$$

$$\frac{\vdash_\phi P \triangleright A}{\vdash_\phi P \triangleright A, x : \tau_{\mathtt{0}}} \qquad \frac{\vdash_{\mathtt{I}} P \triangleright A}{\vdash_{\mathtt{0}} P \triangleright A}$$

As a result, a typed process with the I-mode never has an active output (hence redex) and one with the 0-mode has at most one active output (hence redex). These properties are invariant under reduction.

A type derivation will now correspond to a proof-net with *at most one* $\otimes$-node in each box and at depth 0. If the conclusion of the derivation has mode I, the proof-net does not have any $\otimes$-node at depth 0. Again these properties act locally in respective formalisms' derivations while preserving an exact structural correspondence: we can easily check that the same mutual simulation result as Theorem 1 holds.

5.3. Refined emission determinism

We can further remove the rule $\frac{\vdash_{\mathtt{I}} P \triangleright A}{\vdash_{\mathtt{0}} P \triangleright A}$ from the above, by which a typed process with the 0-mode now has exactly one active output, which is again invariant under reduction. This also corresponds to a well-defined subset of the presented proof-nets: in this case the derivation corresponds to a proof-net with *exactly one* $\otimes$-node in each box. By this constraint a proof-net has one $\otimes$-node at depth 0 if and only if the conclusion of the derivation in the corresponding π-term has mode 0. The number of $\otimes$ at each level controls the number of activities (or threads) in computation.

5.4. Acyclicity

A relation R on a finite set E is *acyclic*[1] if there is no sequence $x_1, \ldots, x_n$ of elements of E with $x_i \; R \; x_{i+1}$ and $x_n \; R \; x_1$.

We add to judgements an acyclic relation on the names appearing in the context: $\vdash_{(\phi)} P \triangleright A; R$.

If $A; R$ is a context with $x \notin A$, we define $x : \tau \; R \; A$ to be the context $A, x : \tau; R'$, where R' is generated from: (1) $R' \restriction_A = R$ ($R' \restriction_A$ means the projection of R' to $\mathsf{dom}(A)$) and (2) for each $y \in \mathsf{dom}(A), x \; R' \; y$. If the resulting R' is not acyclic, we stipulate that $x : \tau \; R \; A$ is not defined.

$$\frac{}{\vdash_{(\mathtt{I})} 0 \triangleright \emptyset; \emptyset} \qquad \frac{\vdash_{(\phi_1)} P_1 \triangleright A_1; R_1 \qquad \vdash_{(\phi_2)} P_2 \triangleright A_2; R_2}{\vdash_{(\phi_1 \boxdot \phi_2)} P_1 | P_2 \triangleright A_1 \odot A_2; R_1 \cup R_2}$$

$$\frac{\vdash_{(\phi)} P \triangleright A, x : \tau_{\mathtt{I}}; R}{\vdash_{(\phi)} \nu x P \triangleright A; R \restriction_A}$$

$$\frac{\vdash_{(\mathtt{0})} P \triangleright A, \vec{y} : \vec{\tau_{\mathtt{0}}}; R \qquad \mathsf{md}(A) = \mathtt{0}}{\vdash_{(\mathtt{I})} !x(\vec{y}).P \triangleright x : (\vec{\tau_{\mathtt{0}}})^! \; R \; A} \qquad \frac{\vdash_{(\mathtt{I})} P \triangleright A, \vec{y} : \vec{\tau_{\mathtt{I}}}; R}{\vdash_{(\mathtt{0})} \bar{x}(\vec{y}) P \triangleright A \odot x : (\vec{\tau_{\mathtt{I}}})^?; R[x/\vec{y}]}$$

$$\frac{\vdash_{(\phi)} P \triangleright A; R}{\vdash_{(\phi)} P \triangleright A, x : \tau_{\mathtt{0}}; R} \qquad \left(\frac{\vdash_{(\mathtt{I})} P \triangleright A; R}{\vdash_{(\mathtt{0})} P \triangleright A; R} \right)$$

A proof-net is *acyclic* if the directed graph obtained by:

- positive edges are oriented upwardly and negative edges are oriented downwardly;
- conclusions of $\otimes$-nodes are removed;
- each !-node with conclusion !N and with n boxes with auxiliary conclusions $\Gamma_1, \ldots, \Gamma_n$ is replaced by a node with an edge !N oriented upwardly (that is towards the !-node) and edges to the ?-nodes under $\Gamma_1, \ldots, \Gamma_n$ oriented downwardly (that is towards the ?-nodes);
- cuts are replaced by directed edges from the corresponding ?-node to the corresponding !-node

is acyclic.

The typing derivations with acyclic relations correspond to acyclic proof-nets. This is because any sub-net of an acyclic proof-net (in the sense above) is acyclic, similarly for a process. The rest follows by induction. Thus we again obtain, by restricting Theorem 1 to these restricted processes/nets, the exact mutual simulation result.

[1] The definition herein gives a different but equivalent presentation of the acyclicity in strongly normalising processes studied in [43].

5.5. Putting everything together

If we put together all the constraints we have stipulated, we get a notion of typed π-calculus which exactly corresponds to proof-nets satisfying the correctness criterion of polarised proof-nets. As a consequence there exist fully complete embeddings of various systems of classical logic into these objects (typed π-terms or correct proof-nets).

The calculus thus obtained (without the refinement in Section 5.3) precisely corresponds to the π^{C}-calculus (the control π-calculus) in [24].

Concerning proof-nets, let us look at the syntax presented in [28]. If we consider correct axiom-free proof-nets, we replace $\flat$-nodes by simple edges, we glue together trees of successive $\otimes$-nodes and 1-nodes (resp. $\parr$-nodes and $\bot$-nodes) and we apply the completion procedure of Section 2, we obtain a proof-net as described here and satisfying all the constraints introduced in this section. Conversely, the proof-nets used here allow for slightly more sharing than in [28] and we have to do a simple unfolding of boxes for !-nodes above a $\otimes$-node. We first introduce a $\flat$-node between each $\otimes$-node and the ?-node below it (inside the box if the $\otimes$-node is inside a box). We replace n-ary $\otimes$-nodes (resp. $\parr$-nodes) by a corresponding tree of binary $\otimes$-nodes and 1-nodes (resp. $\parr$-nodes and $\bot$-nodes) with the appropriate number of leaves. Finally, if we have a !-node whose conclusion is both in a cut and premise of a $\otimes$-node, we make a copy of the !-node together with the (unique) associated box (the auxiliary conclusions of the copy of the box becoming new premises of the corresponding ?-nodes as we did for cut elimination) and we modify the cut so that the conclusion of the new !-node belongs to it instead the original !-node. The proof-net thus obtained belongs to the syntax of [28] and satisfies the associated correctness criterion.

6. Discussions

We discuss some of the possible extensions of the correspondence results discussed in the preceding sections.

Axioms. The axiom for a proposition which may include a variable acts as a generic link in proof-nets. In the π-calculus, such a generic link is realised as a process which sends a datum it receives at one port to another, whose semantic significance is studied in, for example, [23,33]. This is a basic behaviour in communicating systems such as network routers and is called *forwarder* in the study of actor model, cf. [3]. We can directly represent such a generic link using free name passing [7]. Here in order to preserve the preceding translation structure, we add a generic link to the grammar of processes.

$$P ::= \ldots \quad | \quad x \mapsto y$$

whose typing is given by:

$$\frac{}{\vdash_{\mathrm{I}} x \mapsto y \triangleright x : \tau_{\mathrm{I}}, y : \overline{\tau_{\mathrm{I}}};\ (x, y)}$$

$x \mapsto y$ is a non-η-expanded and "polymorphic" version of identity (for example, at type $()^!$, $x \mapsto y$ and $!x.\bar{y}$ are semantically equivalent). For the extended reduction we add two rules. The first rule is:

$$x \mapsto y | C[\vec{x}(\vec{z})P] \searrow x \mapsto y | C[\vec{y}(\vec{z})P]$$

which is also added to the (non-extended) reduction rule when $C[\ ]$ is the trivial identity context $[\ ]$. The second rule:

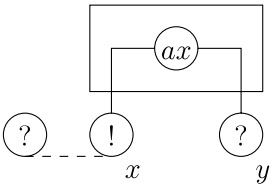
$$\nu x(x \mapsto y) \searrow 0$$

garbage-collects the link when it is no longer necessary.

The corresponding proof-net is built from the standard rule for the axiom link which connects two ends of mutually dual interfaces:

$$\frac{}{\mathsf{P} \qquad \mathsf{P}^{\perp}}$$

by applying:



Polymorphism. We consider parametric polymorphism for the π-calculus in which an existentially typed output not only sends channels but also instantiates types (and, dually, an universally typed input expects both concrete messages and type instantiation). We consider polymorphism where types are explicitly passed at the time of communication, with explicit duality between existential and universal types. The resulting constructions correspond to the polymorphic π-calculus

studied in [41,36] at the level of dynamics and [7] at the level of types. We first extend the grammar of types:

$$\tau_{\mathtt{I}} ::= \ldots \quad | \quad X$$
$$\tau_{\mathtt{O}} ::= \ldots \quad | \quad \bar{X}$$

We set above all type variables $X, Y, \ldots$ are of mode $\mathtt{I}$ and stipulate types of the form $\bar{X}$ have the mode $\mathtt{O}$. Then $\bar{\bar{X}}$ returns X.

The grammar of processes is extended as:

$$P ::= \ldots \quad | \quad !x\Lambda\overrightarrow{X}(\vec{y}).P \quad | \quad \bar{x}\Upsilon\overrightarrow{\tau_I}(\vec{y})P$$

where in the first construct $\overrightarrow{X}$ is a vector of mutually distinct type variables which acts as a binder. The grammar of types now includes quantifications:

$$\tau_{\mathtt{I}} ::= \ldots \quad | \quad (\forall\overrightarrow{X}\,\vec{\tau}_{\mathtt{O}})^!$$
$$\tau_{\mathtt{O}} ::= \ldots \quad | \quad (\exists\overrightarrow{X}\,\vec{\tau}_{\mathtt{I}})^?$$

The duality is extended naturally: for example the dual of $(\forall X\bar{X}(X)^?)^!$ is $(\exists X.X(\bar{X})^!)^?$. The typing rules are standard [7] with $\mathsf{FV}(A)$ standing for the set of type variables in A:

$$\frac{\vdash P \triangleright A, \vec{y} : \vec{\tau}_{\mathtt{O}} \quad \mathsf{md}(A) = \mathtt{O} \quad \{\overrightarrow{X}\} \cap \mathsf{FV}(A) = \emptyset}{\vdash !x\Lambda\overrightarrow{X}(\vec{y}).P \triangleright A \odot x : (\forall\overrightarrow{X}\,\vec{\tau}_{\mathtt{O}})^!}\,\mathrm{in}\forall$$

$$\frac{\vdash P \triangleright A, \vec{y} : \vec{\tau}_{\mathtt{I}}\{\overrightarrow{\tau'_{\mathtt{I}}}/\overrightarrow{X}\} \quad \{\overrightarrow{X}\} \cap \mathsf{FV}(A) = \emptyset}{\vdash \bar{x}\Upsilon\overrightarrow{\tau'_{\mathtt{I}}}(\vec{y})P \triangleright A \odot x : (\exists\overrightarrow{X}\,\vec{\tau}_{\mathtt{I}})^?}\,\mathrm{out}\exists$$

The rules for extended reduction now accompany type instantiation:

$$!x\Lambda\overrightarrow{X}(\vec{y}).P|C[\bar{x}\Upsilon\overrightarrow{\tau_I}Q] \searrow !x\Lambda\overrightarrow{X}(\vec{y}).P|C[\nu\vec{y}(P\{\overrightarrow{\tau_{\mathtt{I}}}/\overrightarrow{X}\}|Q)]$$
$$\nu x!x\Lambda\overrightarrow{X}(\vec{y}).P \searrow \mathtt{0}$$

with only the first rule and a trivial $C[\ ]$ for (non-extended) reduction. This polymorphic extension works with the generic link we introduced in the previous paragraph. The strong normalisation of this fragment is proved in [7].

The corresponding proof-nets add the familiar rules for quantifications for types of the forms $\forall\overrightarrow{X}\,\parr_{1\le i\le n}?\mathsf{P}_i$ and $\exists\overrightarrow{X}\,\otimes_{1\le i\le n}!\mathsf{N}_i$ through new nodes:

$$\frac{\mathsf{P}\{\mathsf{P}'/X\}}{\exists X\mathsf{P}} \qquad \frac{\mathsf{N}}{\forall X\mathsf{N}}.$$

The cut elimination for the proof-nets follows [15,16].

Additives. Additives in Linear Logic present an interactional, fully dualised form of sum types. Their underlying dynamics is closely related with an encoding of a local choice in the π-calculus discussed in [33]. Its typed representation in the π-calculus is discussed in [22] (see also [42,40]) following which we extend the grammar of processes as follows.

$$P ::= \ldots \quad | \quad !x[\&_{i\in I}(\vec{y}_i).P_i] \quad | \quad \bar{x}\mathtt{in}_i(\vec{y})P$$

We call the first form *branching* and the second *selection*. The intuition is that a branching waits with multiple options while a selection selects one of the provided options at the time of interactions.[2] Types are extended as:

$$\tau_{\mathtt{I}} ::= \ldots \quad | \quad [\&_{i\in I}\vec{\tau}_{\mathtt{O}i}]^!$$
$$\tau_{\mathtt{O}} ::= \ldots \quad | \quad [\oplus_{i\in I}\vec{\tau}_{\mathtt{I}i}]^?$$

We type branching and selection as follows.

$$\frac{\cdots \quad \vdash P_i \triangleright A, \vec{y}_i : \vec{\tau}_{\mathtt{O}i} \quad \cdots \qquad \mathsf{md}(A) = \mathtt{O}}{\vdash !x[\&_{i\in I}(\vec{y}_i).P_i] \triangleright A \odot x : [\&_{i\in I}\vec{\tau}_{\mathtt{O}i}]^!} \qquad \frac{\vdash P \triangleright A, \vec{y} : \vec{\tau}_{\mathtt{I}j}}{\vdash \bar{x}\mathtt{in}_j(\vec{y})P \triangleright A \odot x : [\oplus_{i\in I}\vec{\tau}_{\mathtt{I}i}]^?}$$

The extended reduction is given as:

$$!x[\&_{i\in I}(\vec{y}_i).P_i]|C[\bar{x}\mathtt{in}_i(\vec{y})Q] \searrow !x[\&_{i\in I}(\vec{y}_i).P_i]|C[\nu\vec{y}_i(P_i|Q)]$$

together with the garbage collection rule as before, which we omit. The (non-extended) reduction contains only the particular case where $C[\ ]$ is trivial.

[2] At the level of dynamics, these two constructs can be faithfully captured without syntactic extensions, as indicated in [33]: $!x[\&_{1\le i\le n}(\vec{y}_i).P_i]$ becomes $!x(b).\bar{b}(c_1..c_n)\Pi_i c_i(\vec{y}_i).P_i$ and $\bar{x}\mathtt{in}_i(\vec{y})P$ becomes $\bar{x}(b).b(c_1..c_n).\bar{c}_i(\vec{y})P$.

The corresponding deduction in proof-nets are the standard additive rules with boxes [26], with focalised formulae of the forms $\&_{1\leq i\leq n} \parr_{1\leq j\leq i_n} ?\mathsf{P}_{ij}$ (for negative formulae) and $\bigoplus_{1\leq i\leq n} \otimes_{1\leq j\leq i_n} !\mathsf{N}_{ij}$ (for positive formulae). The cut elimination eliminates branches which are not chosen.

We can further extend this fragment with linearity (which, at the level of processes, means simply adding inputs without replication with the corresponding type discipline as in [43], enabling precise representation of state-changing recursive agents [34]; and, at the level of polarised proof-nets, adding the linear lifting following [26]). These extensions reveal a close link which can exist between the systematic articulation of well-behaved processes centring on name passing communication, and the logical structures which arise from the study of distilled proof dynamics.

[*There are strong relations between this work and the work by T. Ehrhard and O. Laurent published in CONCUR'07. However the present work has been completely developed before the Ehrhard–Laurent's one. This is why we do not mention it here. A comparison between the two works is sketched in the introduction of their CONCUR paper.*]

Acknowledgement

The second author was partially funded by the French ANR projet blanc "Curry–Howard pour la Concurrence" CHOCO ANR-07-BLAN-0324.

References

[1] Samson Abramsky, Computational interpretations of linear logic, Theoretical Computer Science 111 (1–2) (1993) 3–57.
[2] Samson Abramsky, Proofs as processes, Theoretical Computer Science 135 (1) (1994) 5–9.
[3] Gul Agha, Actors: A Model of Concurrent Computation in Distributed Systems, MIT Press, Cambridge, MA, USA, 1986.
[4] Samson Abramsky, Radha Jagadeesan, Pasquale Malacaria, Full abstraction for PCF, Information and Computation 163 (2000) 409–470.
[5] Emmanuel Beffara, A concurrent model for linear logic, in: Martín Escardó, Achim Jung, Michael Mislove (Eds.), Proceedings of the 21st Conference on Mathematical Foundations of Programming Semantics, MFPS'05, in: Electronic Notes in Theoretical Computer Science, vol. 155, Elsevier, May 2006, pp. 147–168.
[6] Martin Berger, Kohei Honda, Nobuko Yoshida, Sequentiality and the π-calculus, in: Samson Abramsky (Ed.), Typed Lambda Calculi and Applications'01, in: Lecture Notes in Computer Science, vol. 2044, Springer, May 2001, pp. 29–45.
[7] Martin Berger, Kohei Honda, Nobuko Yoshida, Genericity and the π-calculus, in: FoSSaCs'03, in: Lecture Notes in Computer Science, vol. 2620, 2003.
[8] Emmanuel Beffara, François Maurel, Concurrent nets: A study of prefixing in process calculi, Theoretical Computer Science 356 (2005).
[9] Gérard Boudol, Asynchrony and the pi-calculus, Technical Report 1702, INRIA, 1992.
[10] Gianluigi Bellin, Philip J. Scott, On the pi-calculus and linear logic, Theoretical Computer Science 135 (1) (1994) 11–65.
[11] Jos C.M. Baeten, W. Peter Weijland, Process Algebra, Cambridge University Press, 1990.
[12] Pierre-Louis Curien, Claudia Faggian, L-nets, strategies and proof-nets, in: C.-H. Luke Ong (Ed.), Computer Science Logic, in: Lecture Notes in Computer Science, vol. 3634, European Association for Computer Science Logic, Springer, 2005, pp. 167–183.
[13] Thomas Ehrhard, Laurent Regnier, Differential interaction nets, Theoretical Computer Science 364 (2) (2006) 166–195.
[14] Claudia Faggian, François Maurel, Ludics nets, a game model of concurrent interaction, in: Proceedings of the Twentieth Annual Symposium on Logic in Computer Science, IEEE, IEEE Computer Society Press, Chicago, June 2005, pp. 376–385.
[15] Jean-Yves Girard, Linear logic, Theoretical Computer Science 50 (1987) 1–102.
[16] Jean-Yves Girard, Quantifiers in linear logic II, in: Corsi, Sambin (Eds.), Nuovi problemi della logica e della filosofia della scienza, Bologna, 1991, pp. 79–90, CLUEB.
[17] Jean-Yves Girard, Locus solum: From the rules of logic to the logic of rules, Mathematical Structures in Computer Science 11 (3) (2001) 301–506.
[18] J. Martin, E. Hyland, C.H. Luke Ong, On full abstraction for PCF, Information and Computation 163 (2000) 285–408.
[19] Tony Hoare, Communicating Sequential Processes, Prentice Hall, 1985.
[20] Kohei Honda, Composing processes, in: Proceedings of POPL'96, 1996, pp. 344–357.
[21] Kohei Honda, Mario Tokoro, An object calculus for asynchronous communication, in: Proceedings of ECOOP'91, 1991.
[22] Kohei Honda, Vasco T. Vasconcelos, Makoto Kubo, Language primitives and type disciplines for structured communication-based programming, in: Chris Hankin (Ed.), ESOP'98, in: Lecture Notes in Computer Science, vol. 1381, Springer, 1998, pp. 22–138.
[23] Kohei Honda, Nobuko Yoshida, On reduction-based process semantics, Theoretical Computer Science 151 (1995).
[24] Kohei Honda, Nobuko Yoshida, Martin Berger, Control in the π-calculus, in: Fourth ACM-SIGPLAN Continuation Workshop, CW'04, 2004. Online proceedings.
[25] Ole Jensen, Robin Milner, Bigraphs and mobile processes (revised), Technical report, Cambridge University Computer Laboratory, 2004.
[26] Olivier Laurent, Étude de la polarisation en logique, Ph.D. Thesis, Université Aix-Marseille II, March 2002.
[27] Olivier Laurent, Polarized proof-nets and $\lambda\mu$-calculus, Theoretical Computer Science 290 (1) (2003) 161–188.
[28] Olivier Laurent, Syntax vs. semantics: A polarized approach, Theoretical Computer Science 343 (1–2) (2005) 177–206.
[29] Cosimo Laneve, Joachim Parrow, Björn Victor, Solo diagrams, in: Proceedings of the 4th Conference on Theoretical Aspects of Computer Science, TACS'01, in: Lecture Notes in Computer Science, vol. 2215, Springer, 2001, pp. 127–144.
[30] François Maurel, Nondeterministic light logics and NP-time, in: Martin Hofmann (Ed.), Typed Lambda Calculi and Applications'03, in: Lecture Notes in Computer Science, vol. 2701, Springer, June 2003, pp. 241–255.
[31] Damiano Mazza, Multiport interaction nets and concurrency, in: Martín Abadi, Luca de Alfaro (Eds.), Proceedings of the 16th International Conference on Concurrency Theory, CONCUR 2005, in: Lecture Notes in Computer Science, vol. 3653, Springer, 2005, pp. 21–35.
[32] Robin Milner, Communication and Concurrency, Prentice Hall, 1989.
[33] Robin Milner, Functions as processes, Mathematical Structures in Computer Science 2 (2) (1992) 119–141.
[34] Robin Milner, The polyadic π-calculus: A tutorial, in: Proceedings of the International Summer School on Logic Algebra of Specification, Marktoberdorf, 1992.
[35] Robin Milner, Joachim Parrow, David Walker, A calculus of mobile processes, I/II, Information and Computation 100 (1) (1992) 1–77.
[36] Benjamin C. Pierce, Davide Sangiorgi, Behavioral equivalence in the polymorphic pi-calculus, Journal of ACM 47 (3) (2000) 531–584.
[37] Laurent Regnier, Une équivalence sur les lambda-termes, Theoretical Computer Science 126 (1994) 281–292.
[38] Davide Sangiorgi, πI: A symmetric calculus based on internal mobility, in: TAPSOFT'95, in: Lecture Notes in Computer Science, vol. 915, Springer, 1995, pp. 172–186.
[39] Davide Sangiorgi, The name discipline of uniform receptiveness, in: ICALP'97, in: Lecture Notes in Computer Science, vol. 1256, Springer, 1997, pp. 303–313.

[40] Kaku Takeuchi, Kohei Honda, Makoto Kubo, An interaction-based language and its typing system, in: PARLE'94, in: Lecture Notes in Computer Science, vol. 817, 1994, pp. 398–413.
[41] David N. Turner, The polymorphic pi-calculus: Theory and implementation, Ph.D. thesis, University of Edinburgh, 1995.
[42] Vasco Vasconcelos, Typed concurrent objects, in: Mario Tokoro, Remo Pareschi (Eds.), ECOOP'94, in: Lecture Notes in Computer Science, vol. 821, Springer, 1994, pp. 100–117.
[43] Nobuko Yoshida, Martin Berger, Kohei Honda, Strong normalisation in the π-calculus, Information and Computation 191 (2) (2004) 145–202.